# CompressedBeliefMDPs.jl: A Julia Package for Solving Large POMDPs with Belief Compression

**Logan Mondal Bhamidipaty**[1] and **Mykel J. Kochenderfer**[1]

**1** Stanford University

**DOI:** 10.21105/joss.07346

**Software**

- Review
- Repository
- Archive

**Editor:** Chris Vernon

**Reviewers:**

- @alcap23
- @crvernon

**Submitted:** 18 July 2024
**Published:** 20 June 2025



## Summary

Partially observable Markov decision processes (POMDPs) are a standard mathematical model for sequential decision making under state and outcome uncertainty (Kochenderfer et al., 2022). They commonly feature in reinforcement learning research and have applications spanning medicine (Zhou et al., 2019), sustainability (Wang et al., 2023), and aerospace (Folsom et al., 2021). Unfortunately, real-world POMDPs often require bespoke solutions, because they are too large to be tractable with traditional methods (Madani et al., 2003; Papadimitriou & Tsitsiklis, 1987). Belief compression (Roy et al., 2005) is a general-purpose technique that focuses planning on relevant belief states, thereby making it feasible to solve complex, real-world POMDPs more efficiently.

## Statement of Need

### Research Purpose

CompressedBeliefMDPs.jl is a Julia package (Bezanson et al., 2012) for solving large POMDPs in the POMDPs.jl ecosystem (Egorov et al., 2017) with belief compression (described below). It offers a simple interface for efficiently sampling and compressing beliefs and for constructing and solving belief-state MDPs. The package can be used to benchmark techniques for sampling, compressing, and planning. It can also solve complex POMDPs to support applications in a variety of domains.

### Relation to Prior Work

#### Other Methods for Solving Large POMDPs

While traditional tabular methods like policy and value iteration scale poorly, there are modern methods such as point-based algorithms (Kurniawati et al., 2008; Pineau et al., 2003; Smith & Simmons, 2012; Spaan & Vlassis, 2005) and online planners (Kocsis & Szepesvári, 2006; Ross et al., 2007; Silver & Veness, 2010; Somani et al., 2013; Sunberg & Kochenderfer, 2018) that perform well on real-world POMDPs in practice. Belief compression is an equally powerful but often overlooked alternative that is especially potent when belief is sparse.

CompressedBeliefMDPs.jl is a modular generalization of the original algorithm. It can be used independently or in conjunction with other planners. It also supports *both* continuous and discrete state, action, and observation spaces.

#### Belief Compression

CompressedBeliefMDPs.jl abstracts the belief compression algorithm of Roy et al. (2005) into four steps: sampling, compression, construction, and planning. The `Sampler` abstract type handles belief sampling; the `Compressor` abstract type handles belief compression; the `CompressedBeliefMDP` struct handles constructing the compressed belief-state MDP; and the `CompressedBeliefSolver` and `CompressedBeliefPolicy` structs handle planning in the compressed belief-state MDP.

Our framework is a generalization of the original belief compression algorithm. Roy et al. (2005) uses a heuristic controller for sampling beliefs; exponential family principal component analysis with Poisson loss for compression (Collins et al., 2001); and local approximation value iteration for the base solver. CompressedBeliefMDPs.jl, on the other hand, is a modular framework, meaning that belief compression can be applied with *any* combination of sampler, compressor, and MDP solver.

#### Related Packages

To our knowledge, no prior Julia or Python package implements POMDP belief compression. There are, however, two packages that implement exponential family principal component analysis (EPCA): one in MATLAB (Chambrier, 2016) for Poisson EPCA; the other in Julia for general EPCA (Bhamidipaty et al., 2025). The later API is explicitly designed to be compatible with CompressedBeliefMDPs.jl and may be used to reimplement the original belief compression algorithm in Roy et al. (2005).

# Sampling

The `Sampler` abstract type handles sampling. CompressedBeliefMDPs.jl supports sampling with policy rollouts through `PolicySampler` and `ExplorationSampler` which wrap `Policy` and `ExplorationPolicy` from POMDPs.jl respectively. These objects can be used to collect beliefs with a random or $\epsilon$-greedy policy, for example.

CompressedBeliefMDPs.jl also supports *exploratory belief expansion* on POMDPs with discrete state, action, and observation spaces. Our implementation is an adaptation of Algorithm 21.13 in Kochenderfer et al. (2022). We use $k$-d trees (Bentley, 1975) to efficiently find the furthest belief sample.

# Compression

The `Compressor` abstract type handles compression in CompressedBeliefMDPs.jl. Compressed-BeliefMDPs.jl provides seven off-the-shelf compressors:

1. Principal component analysis (PCA) (Hotelling, 1933),
2. Kernel PCA (Schölkopf et al., 1998),
3. Probabilistic PCA (Tipping & Bishop, 2002),
4. Factor analysis (Thurstone, 1931),
5. Isomap (Tenenbaum et al., 2000),
6. Autoencoder (Kramer, 1991), and
7. Variational auto-encoder (VAE) (Kingma & Welling, 2013).

The first four are supported through MultivariateState.jl; Isomap is supported through ManifoldLearning.jl; and the last two are implemented in Flux.jl (Innes, 2018).

# Compressed Belief-State MDPs

## Definition

First, recall that any POMDP can be viewed as a belief-state MDP (Åström, 1965), where states are beliefs and transitions are belief updates (e.g., with Bayesian or Kalman filters).

Formally, a POMDP is a tuple $\langle S, A, T, R, \Omega, O, \gamma \rangle$, where $S$ is the state space, $A$ is the action space, $T : S \times A \times S \to \mathbb{R}$ is the transition model, $R : S \times A \to \mathbb{R}$ is the reward model, $\Omega$ is the observation space, $O : \Omega \times S \times A \to \mathbb{R}$ is the observation model, and $\gamma \in [0, 1)$ is the discount factor. The POMDP is said to induce the belief-state MDP $\langle B, A, T', R', \gamma \rangle$, where $B$ is the POMDP belief space, $T' : B \times A \times B \to \mathbb{R}$ is the belief update model, and $R' : B \times A \to \mathbb{R}$ is the reward model. $A$ and $\gamma$ remain the same.

We define the corresponding *compressed belief-state MDP* (CBMDP) as $\langle \tilde{B}, A, \tilde{T}, \tilde{R}, \gamma \rangle$ where $\tilde{B}$ is the compressed belief space obtained from the compression $\phi : B \to \tilde{B}$. Then $\tilde{R}(\tilde{b}, a) = R(\phi^{-1}(\tilde{b}), a)$ and $\tilde{T}(\tilde{b}, a, \tilde{b}') = T(\phi^{-1}(\tilde{b}), a, \phi^{-1}(\tilde{b}'))$. When $\phi$ is lossy, $\phi$ may not be invertible. In practice, we circumvent this issue by caching items on a first-come, first-served basis (or under an arbitrary ranking over $B$ if the compression is parallel), so that if $\phi(b_1) = \phi(b_2) = \tilde{b}$ we have $\phi^{-1}(\tilde{b}) = b_1$ if $b_1$ was ranked higher than $b_2$ for $b_1, b_2 \in B$ and $\tilde{b} \in \tilde{B}$.

### Implementation

The `CompressedBeliefMDP` struct contains a `GenerativeBeliefMDP`, a `Compressor`, and a cache $\phi$ that recovers the original belief. The default constructor handles belief sampling, compressor fitting, belief compressing, and cache management. Any POMDPs.jl `Solver` can solve a `CompressedBeliefMDP`.

```julia
using POMDPs, POMDPModels, POMDPTools
using CompressedBeliefMDPs

# construct the CBMDP
pomdp = BabyPOMDP()
sampler = BeliefExpansionSampler(pomdp)
updater = DiscreteUpdater(pomdp)
compressor = PCACompressor(1)
cbmdp = CompressedBeliefMDP(pomdp, sampler, updater, compressor)

# solve the CBMDP
solver = MyMDPSolver()::POMDPs.Solver
policy = solve(solver, cbmdp)
```

## Solvers

`CompressedBeliefSolver` and `CompressedBeliefPolicy` wrap the belief compression pipeline, meaning belief compression can be applied without explicitly constructing a `CompressedBeliefMDP`.

```julia
using POMDPs, POMDPModels, POMDPTools
using CompressedBeliefMDPs

pomdp = BabyPOMDP()
base_solver = MyMDPSolver()
solver = CompressedBeliefSolver(
  pomdp,
  base_solver;
  updater=DiscreteUpdater(pomdp),
  sampler=BeliefExpansionSampler(pomdp),
  compressor=PCACompressor(1),
)
policy = POMDPs.solve(solver, pomdp)  # CompressedBeliefPolicy
s = initialstate(pomdp)
```

```julia
v = value(policy, s)
a = action(policy, s)
```

Following Roy et al. (2005), we use local value approximation as our default base solver, because it bounds the value estimation error (Gordon, 1995).

```julia
using POMDPs, POMDPTools, POMDPModels
using CompressedBeliefMDPs

pomdp = BabyPOMDP()
solver = CompressedBeliefSolver(pomdp)
policy = solve(solver, pomdp)
```

To solve a continuous-space POMDP, simply swap the base solver. More details, examples, and instructions on implementing custom components can be found in the documentation.

# Circular Maze

CompressedBeliefMDPs.jl also includes the Circular Maze POMDP from Roy et al. (2005) and scripts to recreate figures from the original paper. Additional details can be found in the documentation.

```julia
using CompressedBeliefMDPs

n_corridors = 2
corridor_length = 100
pomdp = CircularMaze(n_corridors, corridor_length)
```

# Acknowledgments

We thank Arec Jamgochian, Robert Moss, Dylan Asmar, and Zachary Sunberg for their help and guidance.

# References

Åström, K. J. (1965). Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, *10*(1), 174–205. https://doi.org/10.1016/0022-247X(65)90154-X

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517. https://doi.org/10.1145/361002.361007

Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). *Julia: A fast dynamic language for technical computing*. https://doi.org/10.48550/arXiv.1209.5145

Bhamidipaty, L. M., Kochenderfer, M. J., & Hastie, T. (2025). ExpFamilyPCA.jl: A Julia package for exponential family principal component analysis. *Journal of Open Source Software*, *10*(105), 7403. https://doi.org/10.21105/joss.07403

Chambrier, G. de. (2016). *E-PCA*. https://github.com/gpldecha/e-pca

Collins, M., Dasgupta, S., & Schapire, R. E. (2001). A generalization of principal components analysis to the exponential family. *Advances in Neural Information Processing Systems*. https://doi.org/10.7551/mitpress/1120.003.0084

Egorov, M., Sunberg, Z. N., Balaban, E., Wheeler, T. A., Gupta, J. K., & Kochenderfer, M. J. (2017). POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, *18*(1), 831–835.

Folsom, L., Ono, M., Otsu, K., & Park, H. (2021). Scalable information-theoretic path planning for a rover-helicopter team in uncertain environments. *International Journal of Advanced Robotic Systems*, *18*(2), 1729881421999587. https://doi.org/10.1177/1729881421999587

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In A. Prieditis & S. Russell (Eds.), *Machine Learning* (pp. 261–268). Morgan Kaufmann. https://doi.org/10.1016/B978-1-55860-377-6.50040-2

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*, 498–520. https://doi.org/10.1037/h0070888

Innes, M. (2018). Flux: Elegant machine learning with Julia. *Journal of Open Source Software*. https://doi.org/10.21105/joss.00602

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational Bayes. *arXiv Preprint arXiv:1312.6114*. https://doi.org/10.48550/arXiv.1312.6114

Kochenderfer, M. J., Wheeler, T. A., & Wray, K. H. (2022). *Algorithms for Decision Making*. MIT Press. ISBN: 9780262370233

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *European Conference on Machine Learning*, 282–293. https://doi.org/10.1007/11871842_29

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, *37*(2), 233–243. https://doi.org/10.1002/aic.690370209

Kurniawati, H., Hsu, D. H., & Lee, W. S. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. *Robotics: Science and Systems*. https://doi.org/10.15607/RSS.2008.IV.009

Madani, O., Hanks, S., & Condon, A. (2003). On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, *147*(1), 5–34. https://doi.org/10.1016/S0004-3702(02)00378-8

Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450. https://doi.org/10.1287/moor.12.3.441

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. *International Joint Conference on Artificial Intelligence*, 1025–1030.

Ross, S., Chaib-Draa, B., & others. (2007). AEMS: An anytime online search algorithm for approximate policy refinement in large POMDPs. *IJCAI*, 2592–2598.

Roy, N., Gordon, G., & Thrun, S. (2005). Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research*, *23*, 1–40. https://doi.org/10.1613/jair.1496

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*(5), 1299–1319. https://doi.org/10.1162/089976698300017467

Silver, D., & Veness, J. (2010). Monte-Carlo planning in large POMDPs. *Advances in Neural Information Processing Systems*, *23*.

Smith, T., & Simmons, R. (2012). Point-based POMDP algorithms: Improved analysis and implementation. *arXiv Preprint arXiv:1207.1412*. https://doi.org/10.48550/arXiv.1207.1412

Somani, A., Ye, N., Hsu, D., & Lee, W. S. (2013). DESPOT: Online POMDP planning with regularization. *Advances in Neural Information Processing Systems*, *26*. https://doi.org/10.1613/jair.5328

Spaan, M. T., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for

POMDPs. *Journal of Artificial Intelligence Research*, *24*, 195–220. https://doi.org/10.1613/jair.1659

Sunberg, Z., & Kochenderfer, M. (2018). Online algorithms for POMDPs with continuous state, action, and observation spaces. *International Conference on Automated Planning and Scheduling*, *28*, 259–263. https://doi.org/10.1609/icaps.v28i1.13882

Tenenbaum, J. B., Silva, V. de, & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, *290*(5500), 2319–2323. https://doi.org/10.1126/science.290.5500.2319

Thurstone, L. L. (1931). Multiple factor analysis. *Psychological Review*, *38*(5), 406. https://doi.org/10.1037/h0069792

Tipping, M. E., & Bishop, C. M. (2002). Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society*, *61*(3), 611–622. https://doi.org/10.1111/1467-9868.00196

Wang, Y., Zechner, M., Wen, G., Corso, A. L., Mern, J. M., Kochenderfer, M. J., & Karel Caers, J. (2023). Optimizing Carbon Storage Operations for Long-Term Safety. *arXiv e-Prints*, arXiv:2304.09352. https://doi.org/10.48550/arXiv.2304.09352

Zhou, Z., Kearnes, S., Li, L., Zare, R. N., & Riley, P. (2019). Optimization of molecules via deep reinforcement learning. *Scientific Reports*, *9*(1). https://doi.org/10.1038/s41598-019-47148-x